MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

Artificial Intelligence May 1975

Memo No. 321

A MODEL-DRIVEN GEOMETRY THEOREM PROVER

Shimon Ullman

ABSTRACT

This paper describes a new Geometry Theorem Prover, which was implemented to illuminate some issuses related to the use of models in theorem proving. The paper is divided into three parts: Part 1 describes the G.T.P and presents the ideas embeded in it. It concentrates on the forward search method, and gives two examples of proofs produced that way. Part 2 describes the backward search mechanism, and presents proofs to a sequence of successively harder problems. The last section of the work addresses the notion of similarity in a problem, defines a notion of semantic symmetry, and compares it to Gelernter's concept of syntactic symmetry.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643.

Table of Contents

Introduction:

This paper discusses issues in Geometry Theorem Proving as a case-study in problem solving and in heuristic search. One way of viewing the problem solving process is as a large tree-search: at each step the problem solver chooses one of the strategies available to him/her, and then proceeds to the next decision point. In such a search the mere existence of an algorithmic way for reaching a solution is in many cases of no real interest, since the size of the tree might render it totally impracticable. Rather, the interesting problem is finding intelligent search methods. This means, primarily, using knowledge of the problem domain to direct the search towards the paths with a greater likelihood of success. In this respect the present paper makes a new contribution, as previous G.T.P systems did not address themselves directly to the above aspects of theorem proving.

The G.T.P. system is described in the first two sections of the paper. For methodological reasons the first one concentrates on what is known as the "forward search", while the second discusses the backward (also known as the "analytic") method. This distinction is convenient, although the complete system should be able to make a flexible use of both methods, as well as of some others, like the middle-out search described in the text. The last section of the paper describes a method for dealing with issues of symmetry in geometry problems. In addition to the ability of pre-creating a list of all the symmetries in a problem (the way Gelernter did in his syntactic symmetry method), this new approach enables one to say something like: "This new goal resembles a previous one. So let me check whether the same proof applies here too." Although the treatment of the problem is

somewhat formal, the main intention of this section is to provide a way for dealing with symmetries in a natural and flexible way.

PART I

I.1 General Background

This section is aimed at describing briefly those issues and concepts discussed in previous research which reappear in this work. The first, and the most well known work in this area, was done by Gelernter. He was interested in general aspects of producing formal proofs by machine, and the main ideas he discussed were:

1. The use of an analytic method, also called "chaining backwards", which reasoned backwards from the goal to the hypotheses.
2. The use of the diagram as a filter heuristic, that is, rejecting subgoals which are not consistent with the diagram.
3. Syntactic symmetry: this is an approach to the problem of constructing and justifying a proof, in which one subgoal is proved formally, while another subgoal is said to be "similar" to the first.

Two other systems that produced proofs to geometry theorems were implemented at the M.I.T. AI Lab : The first by Ira Goldstein (Goldstein 1973) and the second by Arthur Nevins (Nevins 1974). One of Goldstein's main concerns was implementing a system in a high level and natural formalism that facilitated the representaion of mathematical knowledge in a program. His PLANNER-based system had indeed a natural and simple structure. But because it depended heavily on rigid top-down tree search, it was doomed to

spend an ever increasing fraction of its time following dead end paths as the problems became more complex. Nevins, on the other hand, explored the other extreme and implemented a system using mostly forward chaining. While Goldstein's system use of the diagram was similar to Gelernter's, Nevins did not use the diagram at all. The next few sections describe the ways the diagram is used in the model-driven G.T.P.

I.2 Description of the System.

The theorem proving system processes a problem in three phases: Phase 1 is a pre-processing; the second stage is an "attach -and-explore" cycle; and phase 3 is a backward search.

I.2a Pre-processing.

The first phase is executed before the hypotheses are given to the system. At this stage the system manipulates the diagram with the ultimate goal of creating what will be called reference-frames. These frames have "slots" in which relevant data are filled in the course of the proof. The following example illustrates our intention. Let (A B C D) be a parallelogram in the diagram. The system creates a frame for the parallelogram, of the following form:

< (A B C D) (segment AB = segment CD) (segment AD = segment BC)

(segment AB parallel segment CD) (segment AD parallel segment BC)

(angle BAD = angle BCD) (angle ABC = angle ADC)

(nil nil nil nil nil nil) >

The current state of knowledge being (nil nil nil nil nil nil) means, that nothing is known yet about this frame. This is an instance of the general structure:

< ;object (e.g. parallelogram)

;important facts (e.g. opposite-sides or angles equality)

;knowledge (e.g. current state of knowledge) >.

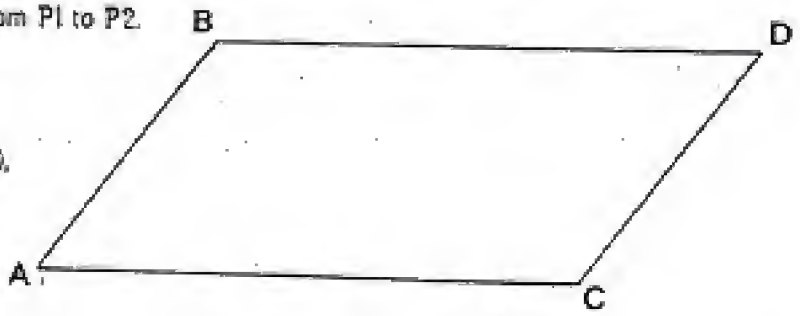
Suppose now that in a later stage the system learns that (segment AD = segment BC). One of the things it will then do, is to attach this fact to the above structure. It will find that the equality (segment AD = segment BC) is the second member of the above frame, and consequently the current state of knowledge will change to: (nil T nil nil nil nil). That is, the second fact attached to the reference frame has been established. The procedure is reversible, in the sense that the T-NIL list, which constitutes the current state of knowledge, can be examined, and the fact that (segment AD = segment BC) be deduced. We say therefore that the fact is attached to the frame. A characteristic feature of that structure is that the same datum might appear (sometimes implicitly) in more than one frame. A fact also appears once explicitly in the general data-base, a list called "KNOWN", that keeps track of the known facts together with their reasons. Thus, KNOWN might contain:

((Segment (A D) = Segment (B C)) (Reason: Given)).

A problem that arises here, is the need for canonic names for geometric objects. This was done using lexicographic order, based on the Lisp ALPHALESSP predicate. (See Nevins, <Nevins 1974> and also <Goldstein 1973> for discussion of this.) Thus a segment (Z X) will be rearranged in the data base as (X Z) and (B A) as (A B). The equality between

those segments will be asserted as: (Segment AB = Segment XZ).

A parallelogram (P1 P2 P3 P4) will not be simply rearranged in ALPHALESSP order,in order to preserve the property that one can go around the parallelogram from P1 to P2 to P3 to P4. Therefore P1, the first point in the canonic name of a parallelogram, will indeed be the ALPHAMIN point (The least point in ALPHALESSP order). P2 will be the ALPHALESSP point between the two neighbors of P1, and then we proceed by going around the parallelogram from P1 to P2. Thus the canonic name of (C D B A) will be (A B D C), rather than (A B C D).



The reference-frames lists created by the system in the first phase include:

1. A list of congruent triangles (CONGRUDLIST)
2. A list of similar triangles (SIMDLIST)
3. A list of parallel lines (PARALLIST)
4. A list of parallelograms (PARADLIST)
5. A list of triangles-with-bisectors (BISECTLIST)

After creating those lists, phase 1 terminates; and phase 2 begins.

1.2b Attach and explore.

In this second phase, the hypotheses are given to the system. The program handles them in a rather simple way, which we might describe as "Attach and explore":

Firstly, it attaches every datum to all its appropriate frames, that is, all the frames that have slots in which the specific datum fits. Note that during this procedure there are no interactions between data: in principle, it can proceed in parallel, if one does not insist on asserting the consequences immediately after they are established.

Secondly, it goes into "Exploration mode". Some exploring functions are invoked, which examine the current state of knowledge of the different frames. If the exploring function finds the list (T nil T nil nil nil) at the end of the parallelogram frame of (A B C D), it asserts the lemma that ((A B C D) is a parallelogram) (Reason: Equal and parallel opposite sides)), adding this new fact to the data-base (The KNOWN list). Recall that the ABCD parallelogram frame was created by examining the diagram, and hence the existence of the frame is not a proof that ABCD is a parallelogram.

1.2c How curious to be?

When new facts are discovered one can proceed by attaching them to their frames, thus creating a new attach-and-explore cycle. This process can iterate until it (hopefully) hits the goal. This is, of course, what we called forward chaining, and examples of proofs produced in this way will be examined later.

Not all data are created equal: Some are always explored; some never are, and some are explored only if the system is curious about them, as will be explained below. If the system knows that in triangle XYZ, (angle XYZ = angle XZY) it concludes that ((segment XY = segment XZ) (Reason: isosceles triangles)), and tries to establish new consequences. On the other hand, if the congruence: (triangle ABC = triangle XYZ) is established, the

consequences:

(Segment AB = Segment XY)

(Segment BC = Segment YZ)

(Segment AC = Segment XZ) etc. are asserted and added to the data base without being further explored. The rationale for this is the following: The statement: (triangle ABC = triangle XYZ) has a different status than the statements:

(segment (A B) = segment (X Y)) or: (angle (A B C) = angle (X Y Z))

It is more powerful than the last two statements and contain more information, (In fact 6 times as much).

The notion "congruence" constitutes therefore a compact concept. Besides containing much information, a concept has to occur frequently enough in order to be a useful compact concept. For example: the concept super-congruent (triangle-1, triangle-2, triangle-3), stating that triangle-1 = triangle-2 = triangle-3, and all are isosceles, is compact, but not useful.

A reasonable place to stop the forward chaining is upon organizing the data in useful, compact concepts. In the geometry domain, "parallelogram" and "congruence" are examples of such concepts. The concept "congruence" for instance, should be represented in such a way, that should some other function search for the segments' equality, it would be immediately available from the congruence. However, if no such requirement is made, this statement will stay still in its compact packing. Thus, we can also make a distinction between active versus passive data.

There are cases, however, when we might wish to alter this situation. The following is an example of such a case: Suppose we are in a backward search, trying to prove: (Angle

ABC = Angle XYZ). We examine our reference-frames to see if there is any natural subgoal, like a pair of diagram-congruent triangles (i.e. a member of CONGRUD), but find none. We would like to tell the system "Make a transition to forward search, and follow the consequences of some fact unexplored so far". The way of making the system more curious about the things it discovers, is by setting the variable "curiosity" to a non-nil value. It is possible to imagine extentions of such an approach that provides a more flexible control over the system's tendency towards forward or backward chaining.

This style of "attach-and-explore" saves future recomputations. For example, suppose that the fact that (Segment AB = Segment XY) is known to the system. It then tries to find a pair of congruent triangles, in which (A B) and (X Y) are sides, and establish their congruence. It eventually fails, because there is not enough data as yet for that. If it does not keep in mind (within the appropriate reference frame) the fact that, say triangles (A B C) and (X Y Z) agree already in one side, it is forced to re-find them and the relevant data about them in the data base, when (Segment BC = Segment YZ) is also known.

1.2d Backward search.

As the first stage of forward search is terminated, the system starts backward chaining, and from now on it might switch from one mode to another. The system's structure was designed to facilitate such a task, but because a more detailed description of that phase is presented in part 2, only general remarks will be made here. The general scheme of the backward search phase is as follows:

When a goal is set forth, the program examines all the possible subgoals. It then

checks whether one of these subgoals has already been established. This situation might arise, when the subgoal is a passive kind of data, in which case the system did not try to establish any new consequences of it. If none of the sub-problems is already proved, the system solicits the help of a Plausible Move Generator (P.M.G). This decision can be efficiently reached because the systems keep records of the current state of knowledge (as opposed to Goldstein's and Nevins' systems, in which this was done only in a limited way).

Another advantage of the current state of knowledge list has been mentioned already, namely, that it can save recomputations. For these reasons, this feature probably has a general applicability and importance.

Suppose the current goal is (segment XY = segment AB). The system is in a position to answer with minimum amount of computation questions like: "Find if (X Y) and (A B) are corresponding sides of some congruent triangles. Examine those triangles to see about which of them we have the largest amount of data, and what kind of data." The system compares the possible subgoals, scores them, and then pursues the highest scored subgoal. (The main function of the P.M.G. is, then, acting as a "Static-Evaluator".) Some modifications of this behaviour are possible:

1. Upon hitting a very high scored subgoal the systems stops evaluating the other subgoals and tries that one. (Similar to Conniver "hang".)
2. As mentioned earlier, if the scores are too low the system might wish to switch to forward search for a while (by setting "CURIOSITY" to T).

1.2e Middle-Out Search:

Upon examination of human methods of proving geometry problems, one observes that a "middle-out search" heuristic is sometimes used, especially in difficult problems. The idea is to find an "interesting" figure in the diagram, like a parallelogram, or a pair of congruent triangles, (those "interesting figures" are closely related to the compact concepts) and try to establish that property. In doing so, the problem solver is relying on the assumption that coincidences are not likely to happen: A parallelogram in the diagram is probably significant and not merely accidental.

Note, that this heuristic (the middle-out search) can also be incorporated quite naturally into the system's structure, for the same reasons that the P.M.G can. Namely, the questions the systems would like to ask while applying the heuristic are easily answered through the use of reference frames.

1.3 Examples.

In this section two examples of forward-chaining proofs produced by the system are presented. The input to the program consists of two kinds of lists: The first contains the Cartesian coordinates of the points, and the other is the list of lines in the diagram. After the system finishes the first stage (phase I), which creates the reference-frames, the "givens" are presented to it. As the proofs were produced using forward search (curiousity = T), the main thing to be inspected is the KNOWN list at the end of the proof. This list contains

all the facts discovered by the system, and therefore it is possible to determine from this list what precentage of the effort was fruitful. The state of knowledge lists of the the different frames at the end shows also, indirectly, what deductions the system had done.

1.3.a Example I.

The first example is Gelernter's second problem which is rather trivial. The only change made, was to introduce another point, O, to complicate the diagram somewhat .

Given:

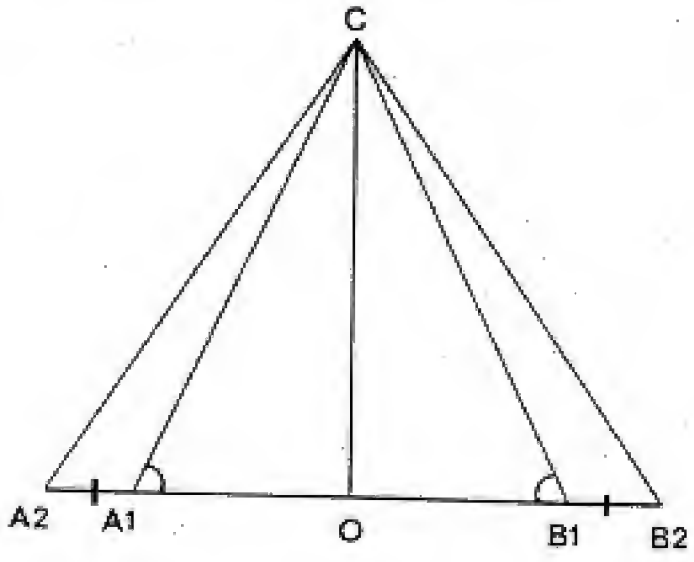
(A1 A2) = (B1 B2)

(Angle (C A1 O) = Angle (C B1 O))

(Line A2 A1 O B1 B2)

Prove:

(C A2) = (C B2)



Under "phase 1" those lists created in that stage are shown. Note that some of the triangles appear twice in the CONGRUD list. The reason for this is, that those triangles are isosceles. Consequently, their vertices can correspond in more than one way.

The way facts are attached to corresponding triangles is different from the T - NIL method described earlier. Rather, each pair of congruent triangles have a numeric value

which gets higher when more is known about the pair, and codes that knowledge. For the sake of simplicity, however, the actual facts known about each pair of triangles (e.g: a. a. s) will be explicitly ststed, instead of the scores.

GELERNTER 2: Given:

1 (Segment (A1 A2) = Segment (B1 B2))

2 (Angle (C A1 O) = Angle (C B1 O))

Goal: (Segment (A2 C) = Segment (B2 C))

Phase 1:

List-of-triangles: ((C B2 B1) (C B2 O) (C B2 A1) (C B2 A2) (C B1 O) (C B1 A1) (C B1 A2) (C O A1) (C O A2) (C A1 A2))

Congruent-triangles:

(((C B2 B1) (C A2 A1)) ((C B2 O) (C A2 O)) ((C B2 A1) (C A2 B1)) ((C B2 A2) (C A2 B2)) ((C B1 O) (C A1 O)) ((C B1 A1) (C A1 B1)))

The Proof:

List of all known facts found in the course of the proof:

((Segment (A1 A2) = Segment (B1 B2)) (Reason: given))

((Angle (C A1 O) = Angle (C B1 O)) (Reason: given))

((Segment (A1 C) = Segment (B1 C)) (Reason: isosceles triangle))

((Triangle (A1 A2 C) = Triangle (B1 B2 C)) (Reason: s a s))

((Angle (A1 A2 C) = Angle (B1 B2 C)) (Reason: triangle (A1 A2 C) = triangle (B1 B2 C)))

((Segment (A2 C) = Segment (B2 C)) (Reason: isosceles triangles))

Q.E.D.

Facts attached to triangles:

((A1 A2 C , B1 B2 C) match in s.a.s)

((A1 B2 C , B1 A2 C) match in s.)

((A1 C O , B1 C O) match in s.s)

((A1 B1 C , B1 A1 C) match in s.s)

List of all the segment-equalities explored:

(((A1 A2) = (B1 B2)) ((A1 C) = (B1 C)))

List of all angle-equalities explored:

(((C A1 O) = (C B1 O)) ((C A1 A2) = C B1 B2)) ((A1 A2 C) = (B1 B2 C)))

The "equivalence-cells":

< (Segments: ((A1 A2) = (B1 B2)) ((A1 C) = (B1 C)))

(Angles: ((C A1 O) = (C B1 O)) ((A1 A2 C) = (B1 B2 C))) >

Comments: 1. The equivalence cells are the way transitivity is handled. All the equal segments, (or angles etc.) are put into the same cell.

2. The system deduced (angle CB1B2 = angle CA1A2) although it did not assert it, for a reason irrelevant here.

3. Some of the facts were not attached to all the possible triangles, because the attaching procedure stopped upon hitting the goal.

1.3.b Example 2.

The second example is Gelernter's fifth problem, which is about as far as Goldstein's system can go. The problem is:

<u>Given:</u>

BC is parallel to AKD.

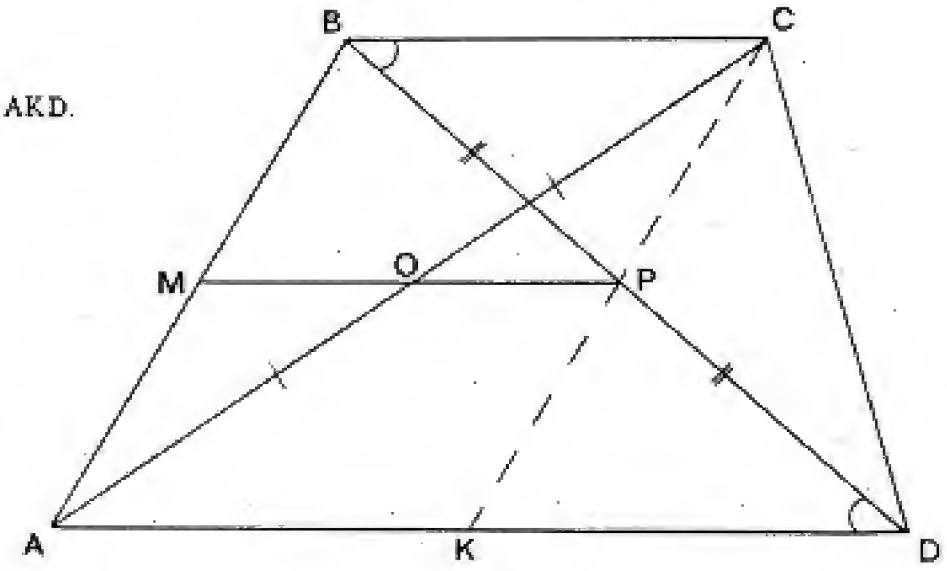
BP = PD.

AO = OC.

(All the lines.)

<u>Prove:</u>

AM = MB



(Actually, line CPK should be a construction.)

And the system's solution is:

<u>KNOWN:</u>

<((LINE (B C) PARALLEL LINE (A K D)) (Reason: given))

((ANGLE (C B D) = ANGLE (A D B)) (Reason: interior alternate angles))

((TRIANGLE (B C P) ~ TRIANGLE (D K P))

(Reason: a a))

((ANGLE (B C P) = ANGLE (D K P)) (Reason: triangle (B C P) ~

triangle (D K P)))

((SEGMENT (B P) = SEGMENT (P D)) (Reason: given))

((TRIANGLE (B C P) = TRIANGLE (D K P)) (Reason: a s a))

((SEGMENT (B C) = SEGMENT (D K)) (Reason: triangle (B C P) = triangle

(D K P)))

((SEGMENT (C P) = SEGMENT (K P)) (Reason: triangle (B C P) = triangle

(D K P)))

((SEGMENT (A O) = SEGMENT (O C)) (Reason: given))

((LINE (A K D) PARALLEL LINE (M O P)) (Reason: line (M O P) is a bisector))

((SEGMENT (A M) = SEGMENT (B M)) (Reason: line (M O P) is a bisector

and parallel to (A K D)) >

Q.E.D

Now let us inspect some of the reference-frames, and the Cells list at the end of the proof.

The list of equivalent cells is:

< (Segments: ((B C) (D K)) ((C P) (K P)) ((B P) (D P)) ((A O) (C O))

((A M) (B M))

(Parallels: ((B C) (M O P) (A K D)))

(Angles: ((C B D) (A D B)) ((B C P) (D K P)) >

Note, that (B C) (M O P), and (A K D) are all in the same parallel cell, therefore they are known to be all parallel.

Parallelograms list: No parallelogram was found. The frame of (A K P M) for example, has one T in its current state of knowlege list, indicating that AK is parallel to MP.

List of triangles with bisectors:

Bisectlist at the end has the following form:

(((D K A) (D P B) ((C P K) (A M B)) (nil T nil))

((A M B) (A O C) ((M O P) (B C)) (T T T))

((B M A) (B P D) ((M O P) (A K D)) (T T T))

((C O A) (C P K) ((M O P) (A K D)) (T T T)))

This means, that the system had found 4 triangles with bisectors in the diagram, and established this relation in 3 cases, using the following bisector theorems:

1. A line bisecting two sides of a triangle is parallel to the third.

2. A line bisecting one side of a triangle and parallel to the base bisects also the second side.

Consequently, the proof produced was short and natural. (Note especially the last two steps.)

Part II

In this part we describe how proofs are produced using the backward chaining mechanism. We shall try to focus on issues of general interest, rather then on details of implementation. Some details are needed, however, to follow the discussion. After presenting the backward search, solutions to a sequence of successively harder problems are discussed.

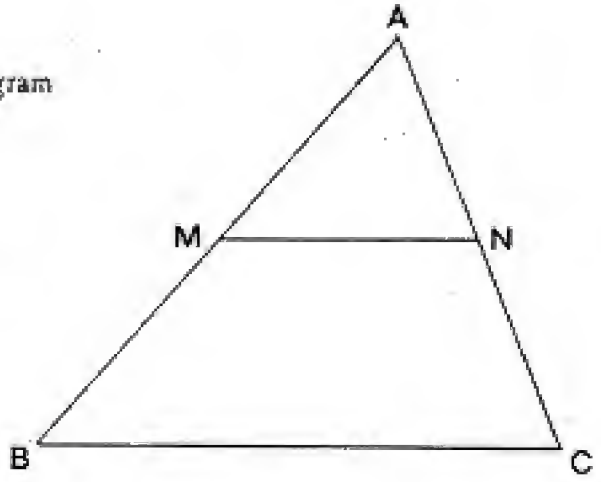
2.1 The experts

The building blocks of the backward search are, using Goldstein's term, the experts. For each possible goal, like segments or angles equality, triangles congruence, bisectors and parallelograms, there is a special structure which connects the goal with the different strategies for proving it. Such a structure, which may be thought of as a branch of the search tree, is called an expert. It consists of a main node, which is the expert's top-level goal, and "daughter-nodes" which are all the possible subgoals, or strategies, for proving the main goal. The experts work hand-in-hand with the P.M.G. (Plausible Move Generator). When a new subgoal is added to the search tree by one of the experts, the P.M.G associates with it a numerical value, called the score of that goal. This score is intended to measure the plausibility of the goal, that is, to give some indication of the a-priori chances to satisfy the goal. Goals are scored according to two criteria: a structural criterion, and a current-state-of-knowledge criterion.

Structural criterion: Suppose the current goal is

(segment AM = segment MB) and the diagram looks like [2.1].

In this case, the subgoal (bisector MN) is added to the search tree with a score of 60, based only on the fact that the structure in the diagram suggests the applicability of that subgoal.



2.1

Current state of knowledge criterion: If, in addition to the structural information, it is also known that MN is indeed parallel to BC, then the score will raise from 60 to 80 points. In the case of congruent triangles, the basic score is 30, and each additional known equality adds 20 points to the score. We summarize the congruence score as 30 + 20k, where k is the number of already known facts.

The scoring scheme was based on subjective impression rather than on accurate computations or experiments. Consequently, the scoring function cannot, in its current state, make the fine distinctions one might, perhaps, want to have. For example, one might argue that the equality of one side and one angle gives a somewhat better support to triangles congruence than the equality of two sides. The rationale is, that in the latter case one of two subgoals is to be established: either the included angle or the third side. While in the first case, there are three possible subgoals, each of which is enough to satisfy the congruence.

2.2 Building a search-tree.

In this section we deal with two problems related to search trees in general, and present the way they were dealt with in the system. The first one has to do with the inefficiency of And-Or trees, and the other with the order of the search.

In the backward chaining, the top level goal constitutes the first node of the search tree. The expert of that goal is then invoked, adding the subgoals it found, as well as their scores, as new branches to the tree. One of the subgoals is then selected as the new current goal, its subgoals are added to the tree, and so on.

The simplest structure that can be created in this way, is known as an "And-Or tree". The name follows from the observation, that it can be represented as a tree in which each goal stems from either an "and" or an "or" node, as shown in figure [2.2]. Such a structure makes it easy to determine whether the top level goal is implied by the proof of a given subgoal. All one has to do is try to "climb" up the tree to the top level goal following the 2 simple rules:

1. An "or" goal is established if at least one of its subgoals is proved.

2. An "and" goal is proved when all its subgoals are.

However, a pure And-Or tree is not very efficient. Consider, for example, the problem of proving a congruence. Even in the case of a simple triangle, (not a right angle or an isosceles one) the produced subtree will have a 13-branches "or", each terminating in a 3-branches "and". [2.2].

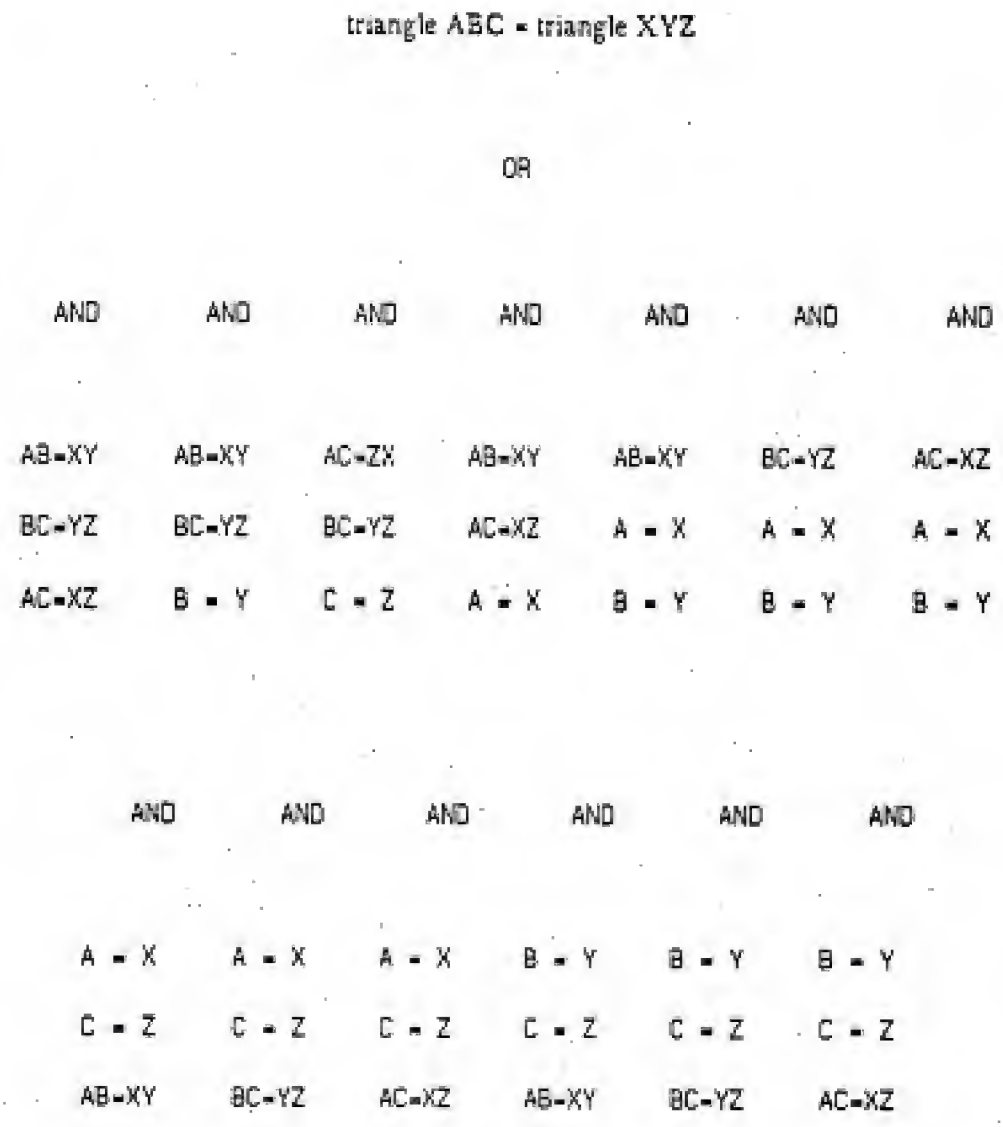


Figure 2.2: The congruence and-or tree

That is, 39 branches, while there are only 6 distinct subgoals: 3 angles and 3 segments equalities.

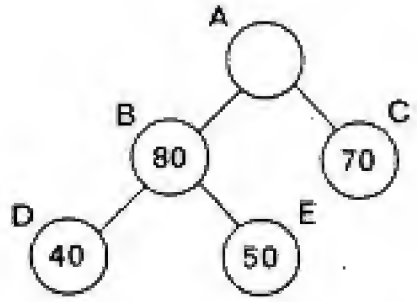
As we already have at hand some forward search mechanism, it is reasonable to make use of it to overcome this difficulty. In addition to "and" and "or" nodes, a new kind of node, called "try-all" was introduced. For example, the goal (triangle ABC = triangle XYZ) can be replaced by the try-all of its 6 different subgoals. All the 6 are then searched, and when one of these equalities is established, it is asserted. The assertion of a goal initiates an attach-and-explore cycle, namely the system switches for a while to forward search, and it watches to see whether the main goal will be deduced from the newly made assertion.

Instead of 39 branches, we now have only 6. The use of the forward mechanism insures that, whenever enough subgoals are established, the main goal will also be deduced.

2.3 The order of the search: Best First.

After attaching a new branch to the search tree, the next decision to be made concerns which of the subgoals is to be selected as the next current goal. The natural candidate is the highest scored subgoal. However, there are two different ways to do this. Consider the following search tree:

Each node represents a goal,
and the numbers are the scores.
We first select B as the highest
subgoal of A. D is the highest
subgoal of B, and will be chosen
if we simply iterate the procedure



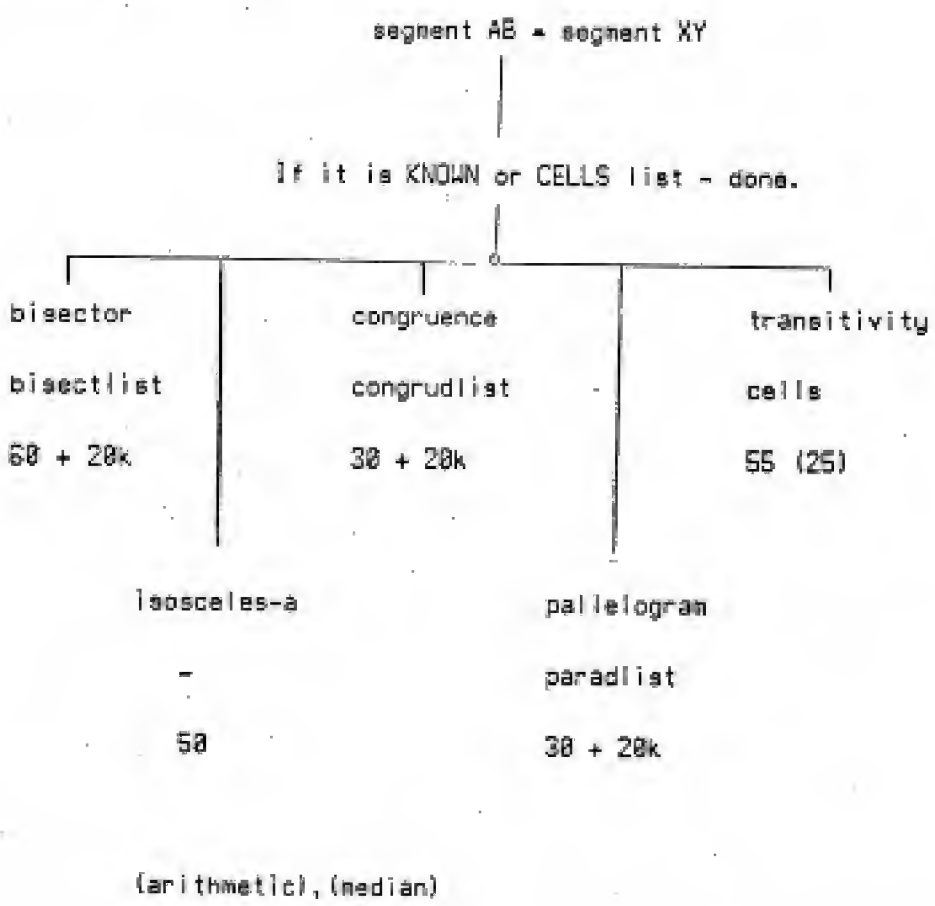
"follow the highest scored subgoal". One might, however, prefer to back up at this stage, choosing C instead, because goal B did not turn out to be as promising as it first appeared, and C is now the highest scored subgoal. This latter approach is the one used by the system. However, it will not back up if the score of C = score of D or E.

To be sure, before a subgoal is added, the subgoals list has to be inspected to verify that it is not there yet. A goal which has no new subgoals gets a 0 score and is never chosen as the current goal.

Keeping track of the tree structure: The internal representation of the search tree is a list of all the subgoals. With each goal 3 numbers are associated: the first is the goal's own identification number. The second is its parent's node number, and the last one is its score. Thus, if MN is a bisector, it might appear in the subgoals list as:((bisector MN) 9 4 60), meaning that it is goal no. 9, a subgoal of goal no. 4, and with a score of 60.

In the following 3 pages, some of the experts are graphically represented.
Each node bears a name, under which, the name of the list the expert inspects, (if any) and the scoring schema are shown.

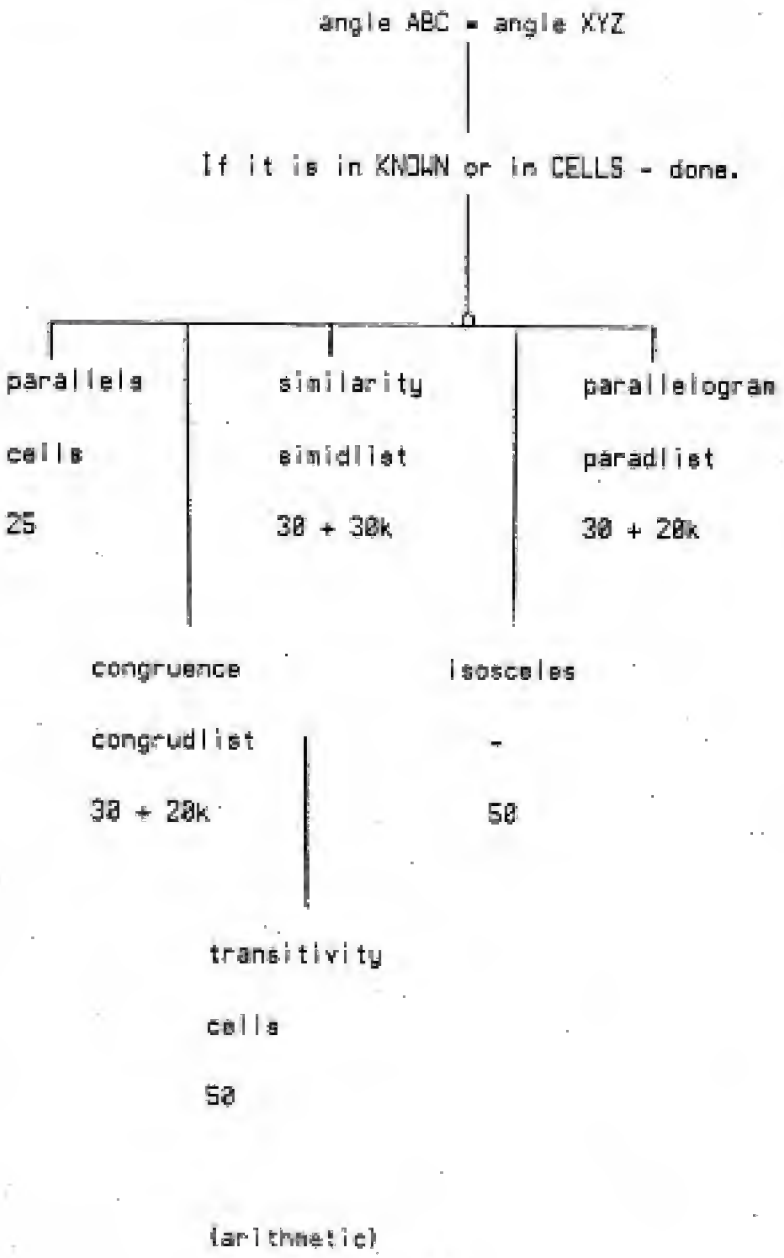
Segment equality expert.



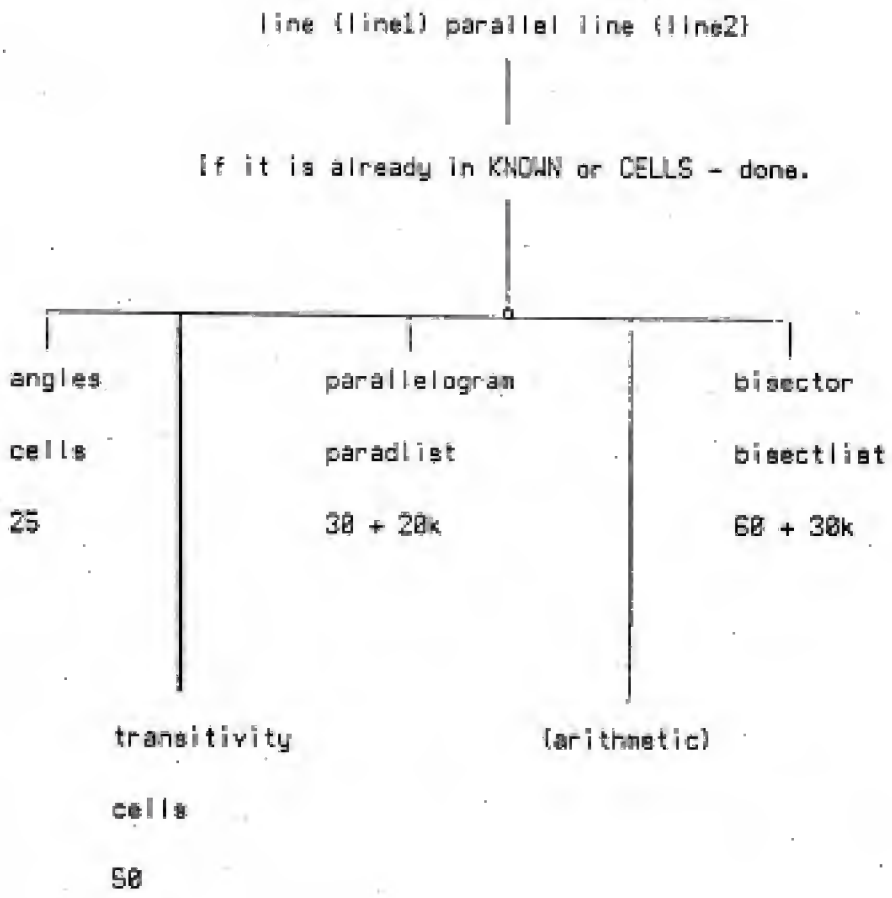
Comments: Isosceles-a means that its subgoal is angles-equality. The angle equality expert has an isosceles-s subgoal, which is later replaced by the appropriate segments-equality goal. Transitivity means finding a segment CD s.t. AB = CD = XY (in the diagram). If CD is already known to be equal to AB or XY, the score is 55, otherwise it is 25.

Segment equality methods based upon theorems involving arithmetic or median are not yet implemented.

Angles equality expert:



Parallel lines expert.



2.4 EXAMPLES.

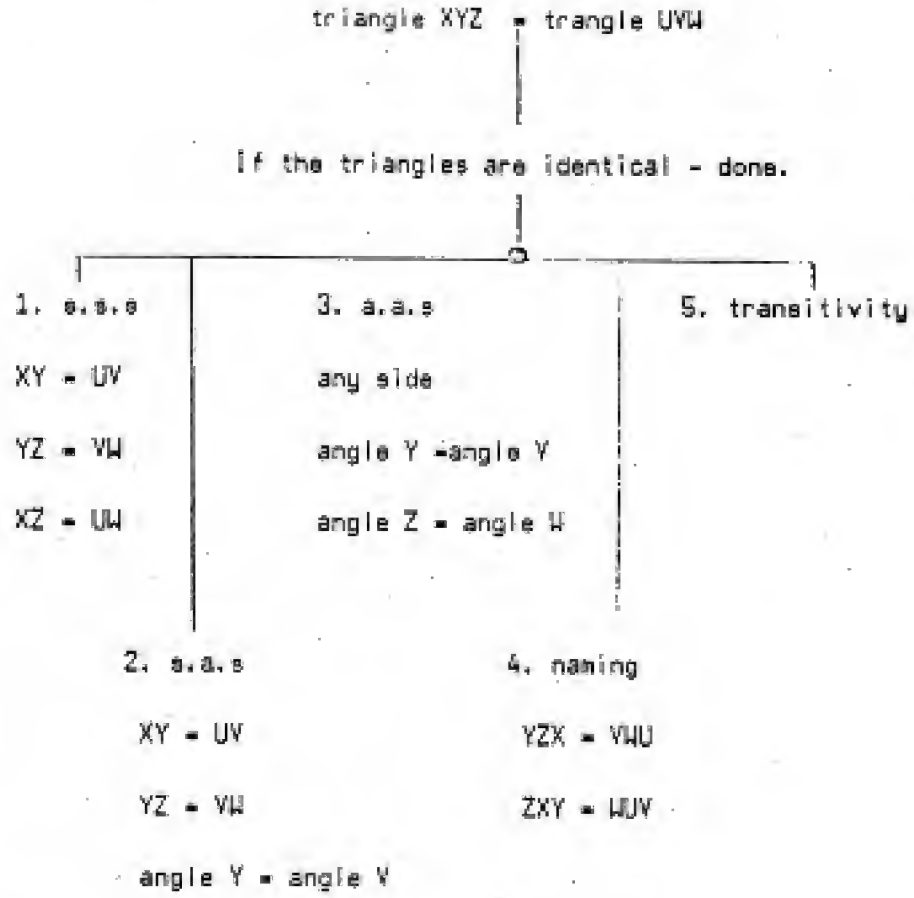
In this section we present solutions to a sequence of successively harder problems, with the goal of getting a more concrete impression of the system's abilities and limitations.

The backward-search part of the system was not implemented as an autonomous theorem prover. Instead, the search algorithm was written as an interactive system. The user has to state his top-level goal, then the system starts asking him questions, to which he responds by typing in the answers. For example: Suppose the goal is: (line line1 is parallel to line line2). the system will present the Paradlist, which is the list of all parallelogram in the diagram, and ask whether the two lines are opposite sides of some parallelogram. The system does most of the bookkeeping, scoring, finding the next subgoal etc. The solution to the third example includes excerpts from such a dialog with the system, while in the other cases only the search-tree will be presented.

2.4a Example 1.

This first example is not really a geometry problem. It is only used to demonstrate the inefficiency that results from a rigid structure of control. Suppose there is a congruence expert, which tries to satisfy its main goal by applying its strategies in a fixed, pre-determined order. Let the order be like in Goldstein's B.T.P. (the version with no P.M.G, and which creates almost pure And-Or trees.)

This expert is illustrated on the next page.



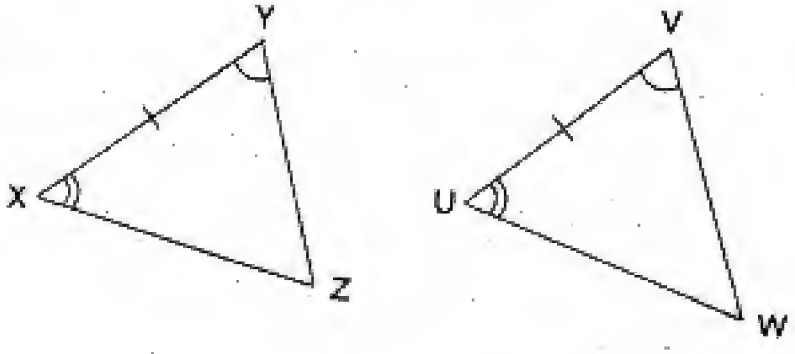
The problem is:

Given:

1. angle ZXY = angle WUV
2. angle XYZ = angle UVW
3. segment XY = segment UV

Prove:

triangle XYZ = triangle UVW



The expert begins by:

Try s.s.s: XY = UV success

YZ = VW failure

Some computation is needed here. The segments equality expert is invoked. As the only subgoal it finds, is (triangle XYZ = triangle UVW), it fails.

Try s.a.s: XY = UV success (already known)

YZ = VW failure

This time it fails immediately, as it remembers past failures.

Try a.a.s: XY = UV success

angle Y =angle V success

angle Z = angle W failure

The angles expert is invoked, and fails. The congruence expert now tries "naming", that is, (triangle YZX = triangle VWU), but fails. It succeeds only with the last strategy, (a.a.s) and the last permutation. (triangle ZXY = triangle WUV).

This is indeed a trivial case, and things will get worse in cases where each subgoal requires a considerable amount of computation. As mentioned in earlier sections, the try-all method, and the incorporation of a P.M.G will help solve this difficulty.

2.4b Example 2.

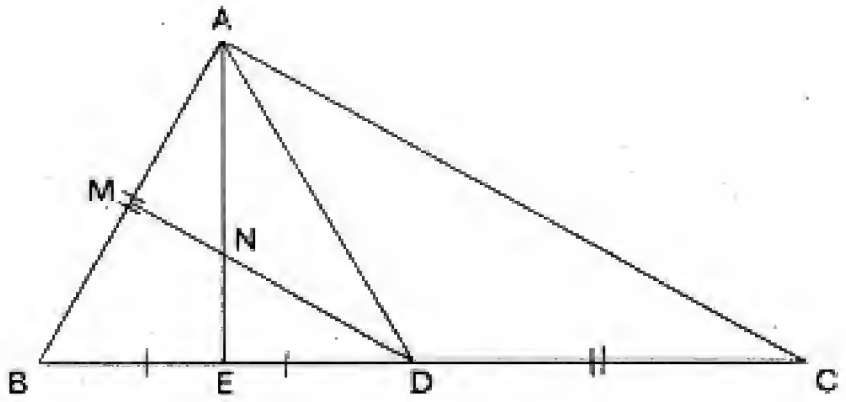
Given: BE = ED

BD = DC

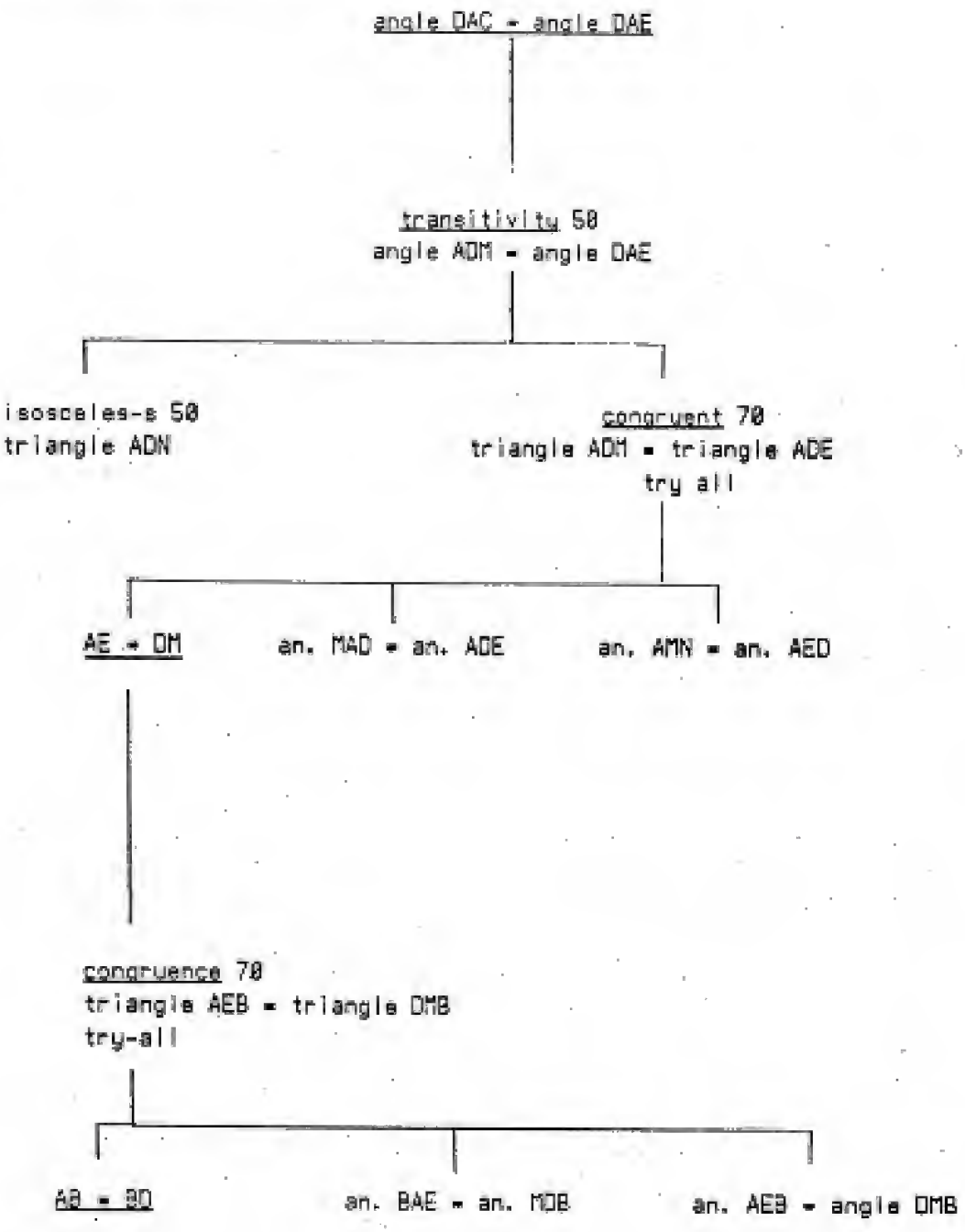
AB = DC

MND parallel to AC

Prove: angle CAD = angle DAE



The complete search-tree is:



Comments: "an." is an abbreviation for "angle".

The underlined subgoals are the ones chosen by the system.

In the forward search phase, the following was discovered:

segment AM = segment MB (reason: bisector)

Consequently, and with the help of the Cells list, the following segments are known to be equal: AM = BM = BE = DE

angle CAD = angle ADM (reason: parallels).

Note, that the above tree is not just the proof tree, but the search tree; all the possible subgoals are presented. Goals that are already established, or that appear higher in the tree, are not considered as new subgoals, therefore some of the "try-alls" have only 3, not 6, subgoals.

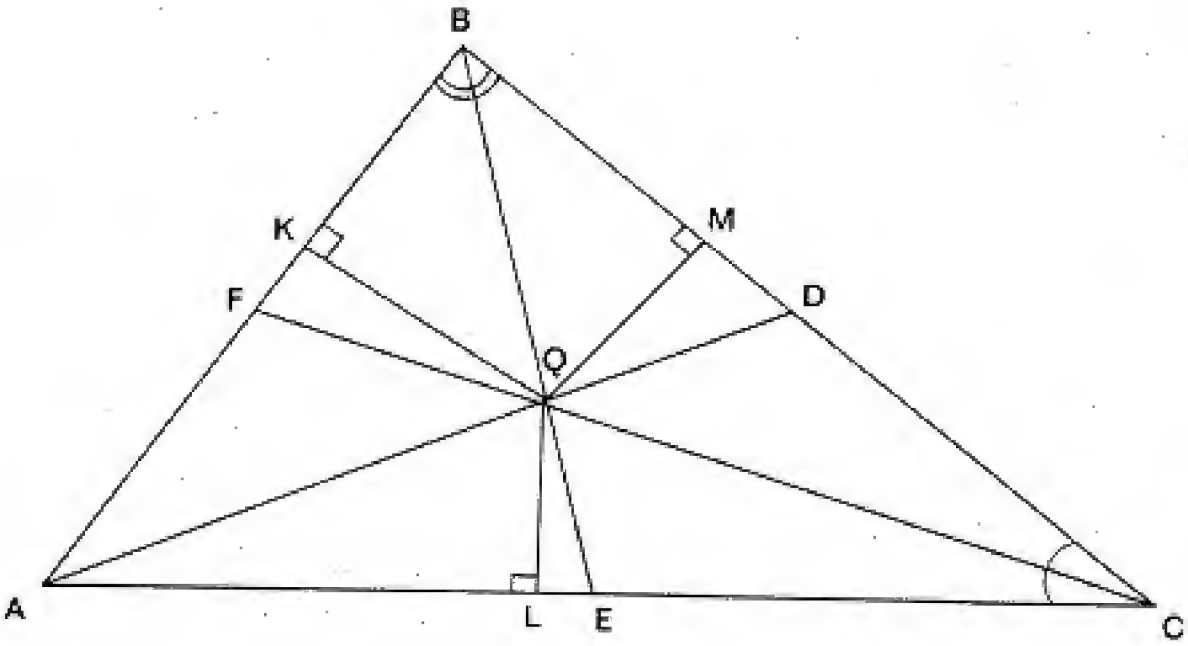
2.4c Example 3.

Given: CQ and BQ are angle bisectors.

QL, QM, QK, are perpendicular to the corresponding sides.

Prove: AQ is also an angle bisector.

(The three angle bisectors of a triangle meet at a point.)



While the previous problem took Nevins' system about 4 seconds to solve, this one involved many more computations, and took about 40 seconds. The reason is, that the backward search mechanism made some fruitless attempts that were time consuming.

To give a better description of all the steps the system must follow till it reaches the solution, the greatest part of the search protocol is also given.

Search protocol, using the interactive system.

(calling angle expert)

(Angle-expert is ready, please type your goal)

=> ((angle KAQ = angle LAQ) 0 0 0)

Comment: As explained in 2.3, (page 25) (0 0 0) indicates that this is the top level goal.

(Is the goal in the Known or Cells lists? the Cells list is:)

<(parallels:)

(segments: (cm = cl) (bm = bk) (ql = qm = qk))

(angles: (dcq = ecq) (mbq = kbq) (cqm = cql) (bqk = bqm) (qka = qkb = qla = qlc = qmb = qmc))>

=> no.

(Try parallels: are the angles known to be equal as a consequence of already established parallel lines? (search parallels in the Cells list))

=> no.

(Try isosceles: are they in an isosceles triangle?)

=> no.

(try congruence: are they corresponding angles of a pair of congruent triangles in the diagram? the Congrudlist is:)

<(a k q) = (a l q) (c l q) = (c m q) (b k q) = (b m q)>

=> yes.

(Type the congruent triangles.)

=> (AKQ ALQ)

(score = 30 + 20k please type the score)

Comment: k is the number of known equalities, as explained in 2.3

=> 70

((congruent AKQ ALQ) 1 0 70)

(Try similarity: are they corresponding angles in similar triangles? Simidlist is:)

nil

=> no.

(Try parallelograms: are they opposite angles of a parallelogram? Paradlist is:)

nil

=> no

(Try transitivity: check whether at least one of the two angles appears in cells)

(Cells list is displayed again)

=> no.

(Search the parallelist. Is there a pair of parallel lines that might imply the angles equality? parallelist is:)

nil

=> no.

(end of angles-expert)

The subgoals list at this stage is:

(((angle kaq = angle laq) 0 0 0) ((congruent akq alq) 1 0 70))

(calling congruence-expert)

(Congruence expert is ready. Please print your goal)

=> *((triangle AKQ =triangle ALQ) 1 0 70)*

(Type in the first triangle)

=> *(A K Q)*

(Type in the second one)

=> *(A L Q)*

(I will now print out the possible subgoals. If the subgoal appeared already in "subgoals" print 'no'. If it is already in Known print 't'. Otherwise print 'yes', indicating that you accept it as a subgoal)

(ak = al)

=> *yes.*

(kq = lq)

=> *yes.*

(aq = aq)

=> *T.*

(angle AQK = angle AQL)

=> *yes.*

(angle QLA = angle QKA)

=> *T*

(As it happens to be already in Known.)

(end of congruence expert. Expand now all the try-alls)

the subgoals list at this stage is:

<((angle kaq = angle laq) 0 0 0)

((congruent akq alq) 1 0 70)

try-all: ((ak = al) 2 1 yes) ((kq = lq) 3 1 yes)

((aq = aq) 4 1 t) ((angle aqk = angle aql) 5 1 yes)

((angle qla = angle qka) 6 1 t) end-try-all)>

Comment: In the case of "try-all" all the subgoals are to be expanded. This is indicated in the system by the flag "yes" replacing the numeric score.

(calling segments-expert.)

(segment-expert is ready, print your goal.)

((segment kq = segment lq) 3 1 yes)

(is it in known or in cells? the cells list is:)

<(parallels:)

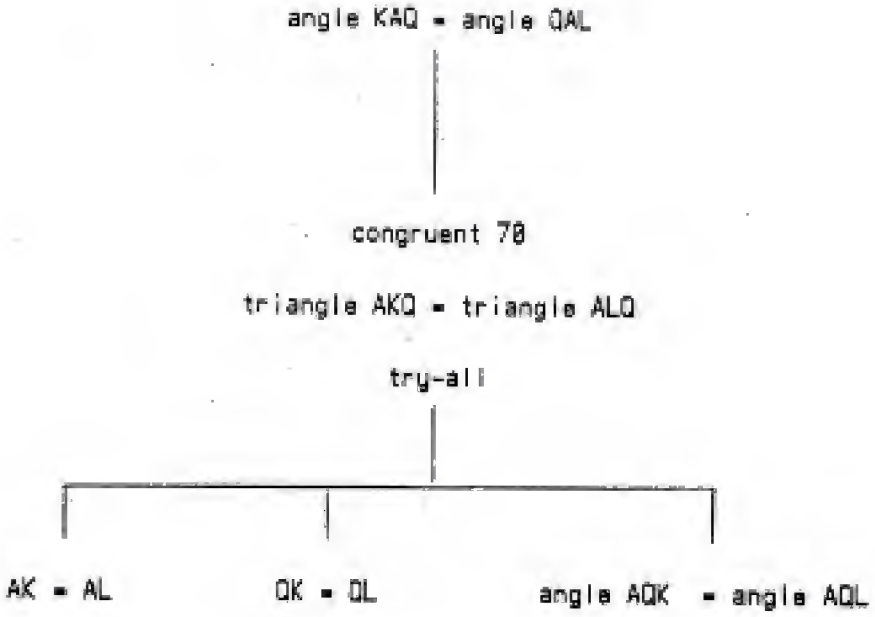
(segments: (cm = cl) (bm = bk) (ql = qm = qk))

(angles: (dcq = ecq) (mbq = (kbq) (cqm = cql) (bqk = bqm) (qka = qkb = qla = qlc = qmb = qmc))>

=> *yes*

(subgoal proved)

The entire search tree is:



Comments: Subgoals which are known to be true, like (angle QKA =angle QLA) are not shown as subgoals.

In the forward phase, the following was established:

triangle CMQ = triangle CLQ

triangle BMQ = triangle BKQ

The search was quite short, so that this problem should not be more difficult for the system than the previous one.

2.4d Example 4.

This is the most difficult problem solved by Nevins' system.

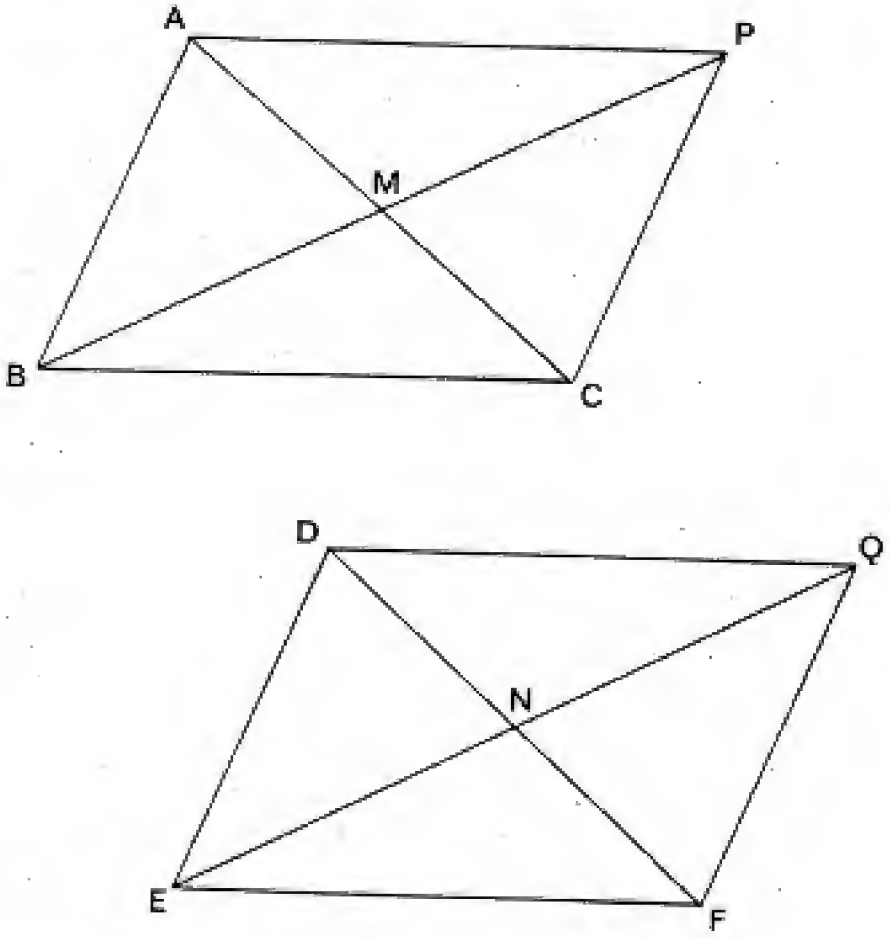
Given: AM = MC DN = NF Prove: AC = DF

BM = MP EN = NQ

AB = DE BC = EF

BM = EN

Diagram for problem 4.



In the forward search, the system discoveres that ABCP and EDQF are parallelograms. It also discovers all the congruent triangles within each parallelogram. The backward search tree is larger this time, threfore it will be listed rather than shown

graphically.

Backward search: Top level goal is AC = DF

The goal has two subgoals:

1. triangle ABC = triangle DEF, score = 70. (Two sides, equalities already known).

2. triangle ACP = triangle DFQ, score = 70. (Two sides, found in "cells")

The first one is chosen, and replaced by its "try-all"s:

3. angle ABC = angle DEF.

4. angle BAC = angle EDF.

5. angle ACB = angle DFE.

(The other subgoals are either known or previous goals.)

Now, the "try-all"s are searched and scored.

Subgoals of 3:

6. Transitivly, score = 50.

Subgoals of 4:

7. triangle ABM = triangle DEN, score = 70.

8. Transitivity, score = 50.

Subgoals of 5:

9. triangle BMC = triangle ENF, score = 70.

The system now chooses goal 7. It does not back up to an equal-score subgoal.

Subgoals of 7 are the "try-all"s:

10. angle ABM = angle DEN.

11. angle AMB = angle DNE.

The system also tries AM = DN, but this is not really necessary. It does not deal with arithmetic, leaving it to the forward search. But if it did, AM = DN would have been one of the first subgoals, and therefore not considered here.

Subgoal of 10 is:

12. triangle ABP = triangle DEQ, score = 70.

And of 11 is:

13. Transitivity, score = 50.

When 12 is picked, AP = DQ is found in Cells and BP = EQ by arithmetic. When they are asserted, a forward search cycle is initiated, and the top level goal is also deduced.

Similarity is used only when two triangles which are similar, but not congruent are found. The rationale is, that the system might as well try to establish congruence. If it discoveres that they agree in 2 angles, similarity will be deduced anyway.

The main difficulty in this problem is the symmetry involved. Symmetry is also the reason that the system chooses a current subgoal in the case that both it and a previous goal have the same score. In problems with much symmetry many subgoals at the same level might have the same score, and it is not advisable to explore all of them. Note also, that because of that the system is not geared toward finding always the shortest proof. To produce the shortest proofs, it should follow the "best first" rule, but also have the ability to deal with symmetries.

2.5 Difficult problems.

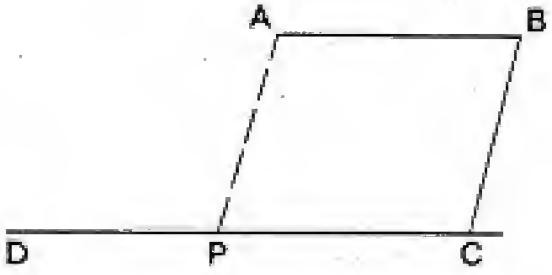
The issue of solving difficult problems is closely related to some other topics often raised in A.I research, such as abstraction, planning and learning. Upon attacking a problem, especially a hard one, one often begins by generating some abstract plan, and it is not until a later stage, that one worries about filling in all the details. The ability to analyze the problem in not-too-detailed terms is, at that stage, of great importance. Abstraction also plays an important role in learning. For example, a crucial problem in the geometry learning process is the extraction and representation of the new knowledge gained in the course of finding a proof to a theorem.

In this section, the system's features that enable it to generate some plans are discussed. In the particular domain of plane geometry, the most difficult problems are those which require clever constructions. The question of incorporaing construction heuristics in a geometry theorem prover was addressd by R. Wong in his M.Sc. thesis, <Wong 72> and here are two of his heuristics.

1. situation: AB is parallel to CD.

AB is not equal to CD.

goal: AB + XY = CD + UV.



construction:

line AP parallel to BC.

2. situation: angle ABY = angle CBX.

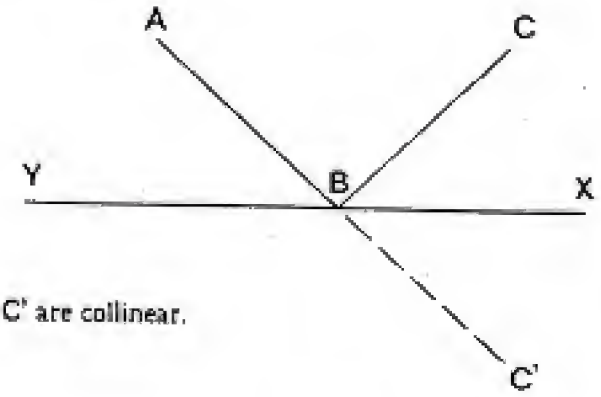
A and C are on the

same side of XY.

goal: AB + BC = some PQ.

<u>construction:</u>

segment BC' s.t. BC = BC', and A B C' are collinear.



Note, that in checking the applicability of a goal, certain structural relations are to be verified. In Wong's heuristics, relations like "same side" or "inside" are often to be determined.

But the role of the diagram does not end here. Even preconditions like segment or angle equalities are often verified only in the diagram (as opposed to being proved), following which a construction is made, and only then the formal proof follows. Take for example the condition "AB is not equal to CD"; one is not supposed to <u>prove</u> this statement before applying the construction, but rather, to verify it in the diagram. It seems plausible, therefore, that if a system is to have powerful constructive capabilities, it should include semantic processes (that is, analysis of the diagram) interacting with its other parts. In this respect the approach represented in the described system might be a step in the right direction.

The diagram pre-processing and the creation of the reference-frames, are also useful for getting a global picture of the problem. That is because they enable the system to notice the important relations in the problem, and organize them in compact concepts like

parallelograms, bisectors or congruent triangles. To make this point clearer, a "plan" was produced with the help of the system, by using the following procedure:

1. Make a backward search from the top level goal to depth 2.

2. State the 2 highest scored subgoals in each of these 2 levels.

3. Examine the different frames, find the highest scored one. (middle search).

Return, for example, to example 2, and examine the produced plan. In level 1 there was only one subgoal, transitivity. In level 2 there were 2 subgoals, congruence and isosceles. The highest scored frame is (triangle ABE = triangle DBM) (the bisector is not to be counted, because it had been already proved). Now put all this into a more human-like form, and the result is something like:

"In order to show that angle DAN = angle CAD, I have to prove that angle DAN is equal to angle ADN. To do this, I can either establish the congruence of triangles ADM and AED, or that AND is isosceles. I'll try the congruence first. It also seems plausible that triangle ABE = triangle ADM, so I might try to establish and use that".

The above procedure is not intended to work as a plan-generator, but rather to show that due to the diagram processing stage the system acquires some capability of attacking the planning problem in a rather natural way.

PART III SEMANTIC AND SYNTACTIC SYMMETRY.

One of Gelernter's primary concerns in his original work on Geometry Theorem Proving was the problem of symmetry in mathematical proofs. The problem is to clarify when, and on what grounds, a mathematician is justified in giving only one part of a proof, and claiming that the other part is done in a similar, or symmetric, way. Gelernter had implemented an algorithm that was executed before the beginning of the proof search, producing all possible symmetries in the problem. Then, when the system was about to establish some subgoal, it first examined the list of all symmetries to check whether a similar subgoal had already been proved or searched. His purely syntactic definition of symmetry will be explained in the next paragraph. We will, however be interested in a somewhat different problem. Rather than pre-creating all possible symmetries we shall address the issue of similarity between two given goals. In section 3 we shall see how this problem can be solved in our system. The solution will be based on the notion of semantic rather than syntactic symmetry (The term "semantic" indicates that the diagram is used.)

Syntactic symmetry .

The definition of symmetry embodies the idea that two goals are similar if it is possible to prove one of them by re-naming the variables in the proof of the other.

Let H = (H1 ∧ H2 ∧ ... ...∧ Hn) be the conjunction of all our hypotheses, GOAL1 be our goal, and α be the set of all variables appearing in H. Another goal,

GOAL2 is said to be similar or symmetric to GOAL1 if there exist a permutation π of the variables α such that:

1. GOAL1 ($\pi\alpha$) = GOAL2

2. H ($\pi\alpha$) = H

where $\pi\alpha$ is the permutation π of the variables α.

Example: Suppose we are given the following:

1. Segmet (A B) = Segment (A1 B1)

2. Segment (A C) = Segment (A1 C1)

3. Segment (B C) = Segment (B1 C1)

and GOAL1 is (Angle (A B C) = Angle (A1 B1 C1)). or If we are also given:

1a. Segment (X Y) = Segment (X1 Y1)

2a. Segment (X Z) = Segment (X1 Z1)

3a. Segment (Y Z) = Segment (Y1 Z1)

and GOAL2 is (Angle (X Y Z) = Angle (X1 Y1 Z1)), we can say, according to the above definition, that the two goals are similar under the permutation P1:

X <- A Y <- B Z <- C

X1 <- A1 Y1 <- B1 Z1 <- C1

However, goal: (Angle (X Z Y) = Angle (X1 Y1 Z1)) is not to be considered similar to GOAL1.

What Gelernter's algorithm did, was to produce all the permutations π such that H($\pi\alpha$) = H. Then, if the system sets up a subgoal SUBGOAL1, it can search whether it had already proved or searched the same goal under different variable naming.

Semantic Symmetry.

For the purpose of the current section we shall slightly change the format of the reference frames described in part I. Each frame will be a list whose first element is a geometric figure and the rest are all the relations between pairs of its elements, all of them in canonic form. The frame of parallelogram (A B C D) will thus be:

< (A B C D) (AB = CD) (AC = BD) (AB Parallel CD) (AC Parallel BD)

(Angle DAB = Angle BCD) (Angle ADC = Angle ABC) >

Some definitions:

NEIGHBOR of a goal GOALI is any goal that is found in a frame in which GOALI is also found. In the above example of a parallelogram frame (AB = CD) will be a neighbor of (AC Parallel BD).

The frames are to be constructed in such a way, that if there is a rule of inference in the system of the form (Q1 ∧ Q2 ... ...∧Qn ⊃ GOAL1) , then Q1 to Qn are all neighbors of GOALI in (at least) one of the frames. Therefore, if the system knew the rule that equality of diagonals in a quadrilateral implies that the quadrilateral is a parallelogram, the relation between the diagonals, or (AC = BD) in our example, should have been added to the parallelogram frame.

SAME-POSITION (GOALI GOAL2): Two goals GOALI and GOAL2 are said to be in the same position in the frames if for each frame GOALI is in, there is a frame in the same list in which GOAL2 is a member, and vice versa. Note that one of the lists is the KNOWN list, the list of all statements established so far. Therefore Same-position (GOALI

GOAL2) implies, among other things, that GOAL1 has already been established if and only if GOAL2 has been. Informally, (AB = CD) and (XY = UV) are in the SAME-POSITION, if they are members of similar geometric figures, and if we have about them the same amount of information. This is, however, not enough: we wish to assert that their neighbors, which are their possible antecedents in the proof, are also in the same-position. For this end the EQUIVALENCE relation is defined.

EQUIVALENCE: Two goals, GOAL1 and GOAL2, are said to be equivalent of order 0 (GOAL1 $\overset{0}{\sim}$ GOAL2) if there exists a permutation π such that:

a. GOAL1 ($\pi\alpha$) = GOAL2.

b. Same-position (GOAL1, GOAL2).

GOAL1 is equivalent of order n to GOAL2 (GOAL1 $\overset{n}{\sim}$ GOAL2) if there exist a permutation π such that:

a. GOAL1 ($\pi\alpha$) = GOAL2.

b. For each Pi, a neighbor of GOAL1, there is a P'i, a neighbor of GOAL2, such that Pi $\overset{n-1}{\sim}$ P'i. (Under the same permutation.)

GOAL1 is equivalent to GOAL2 if they are equivalent of any order.

Lemma:

If GOAL1 has a proof of length n, and GOAL2 $\overset{n}{\sim}$ GOAL1, then GOAL2 as well has a proof of length n. (Which is the same proof as that of GOAL1, with the permuted variables).

Proof:

By induction. For n = 0: GOAL1 has a proof of length 0 only if it is in the KNOWN list. GOAL1 $\overset{\emptyset}{\sim}$ GOAL2 implies that they are in the SAME-POSITION, therefore GOAL2 is also in the KNOWN list and therefore has a 0 - length proof. The induction step:

Let GOAL1 $\overset{n+1}{\sim}$ GOAL2, and GOAL1 has some proof of length n+1. The last step of this proof is of the form: P1 ∧ P2 ∧... ...∧ Pn ⊃ GOAL1. From the definition of equivalence, there exist a permutation π such that P1(πα) = P'1 P2(πα) = P'2,... ...,Pn(πα) = P'n. when P'1... ...P'n are all neighbors of GOAL2, and, by the induction hypothesis, all have proofs of length n.

P'1 ∧ P'2 ... ...∧ P'n ⊃ GOAL2 will be the n+1'th step in the proof of GOAL2.

Corollary:

If GOAL1 ~ GOAL2, then GOAL1 is provable if and only if GOAL2 is and the proofs are the same, except for the variables naming.

Comparing Semantic and Syntactic Symmetries.

One basic idea was common to the two notions: two proofs are symmetric if they have exactly the same structure but deal, perhaps, with a different set of points.

There are, however, two significant differences:

1. Incorporation of semantic predicates, thereby constraining the possible permutations.

By checking that SAME-POSITION (GOAL1 GOAL2) (a predicate which cannot be used in a proof) we verify things like: Both (AB CD) and (XY UV) are corresponding sides of of CONGRUD (congruent by the diagram) triangles. If this test fails, (AB = CD)

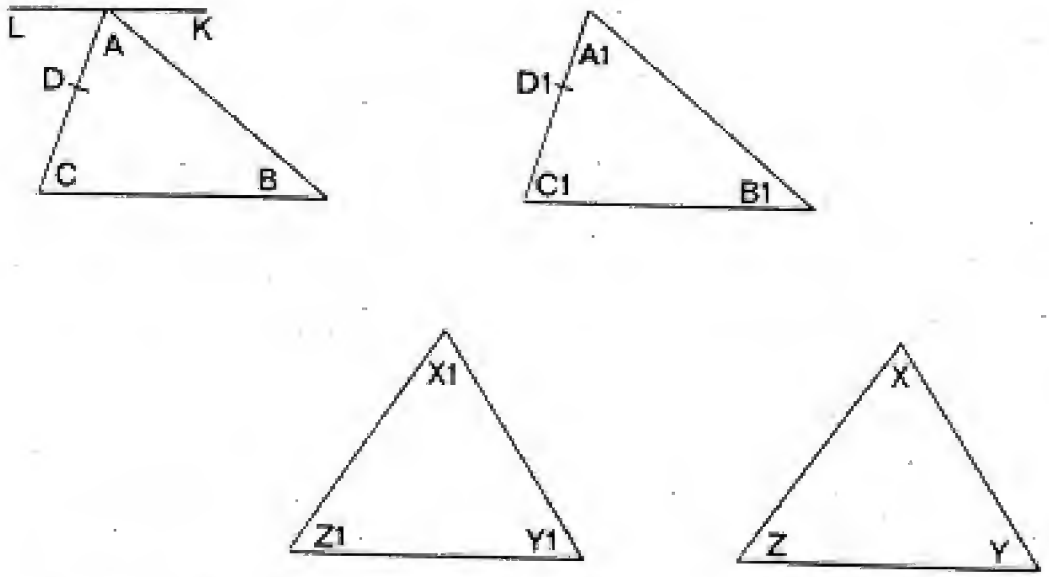
and (XY = UV) are not to be considered symmetric goals.

2. Restriction of the requirements to the relevant points. In the syntactic symmetry definition it was required that the conjunction of all the hypotheses will transform into an equivalent sentence. In the semantic symmetry, on the other hand, we inspected only those predicates which are connected to the goals through some chain of neighbors.

The two differences stated above have the following implications:

A. Efficiency of the search.

The main step in showing symmetry is finding the right permutation. Consider the following diagram and set of predicates:



Segment (A B) = Segment (A1 B1)

Angle (D A B) = Angle (D1 A1 B1)

Angle (A B C) = Angle (A1 B1 C1)

Line (A D C) ; Line (A1 D1 C1)

The goal is to show Segment (A C) = Segment (A1 C1).

If goal2 is to prove Segment (X Z) = Segment (X1 Z1), we know from the requirement that goal1(σπ) = goal2, some constraints on the permutation. A possible such permutation is: A is transformed to X, C -> Z, A1 -> X1, C1 -> Y1. But to what does D transform? (remember the diagram is not to be used).

Proceeding on syntactic grounds, we notice that D appears in the predicate LINE (A D C). Therefore we look for a predicate LINE (X ?Y Z), and the value of ?Y will be a possible candidate for πD. The problem is that we might have many such points satisfying LINE (X ?Y Z). The use of semantic predicates , namely predicates about the diagram, will limit the number of candidates in the same way that it does in the proof search. Suppose we have for example a predicate CONGRUD (A B C X Y Z) which is TRUE if triangle (A B C) is congruent in the diagram to triangle (X Y Z). The number of points satisfying CONGRUD (A B C X Y ?Z) should be quite smaller then LINE (X Y ?Z).

B. Weakening a too-strong requirement.

The requirement that H (πσ) = H, when H is the conjunction of all the hypotheses is too strong. Suppose we have [figure 3.1] a predicate (between A K L), stating that A is between K and L, as part of the given. In that case, the transformatoin of A to X makes the predicate false, (and there are no other candidates for K and L). Consequently GOAL2 is not to be considered syntactically symmetric to GOAL1. However, as there is no chain of neighbors from the goals to the predicate (between A K L) , GOAL2 is still semantically symmetric to GOAL1.

Finally, the question of incorporating the notion of symmetry into the system is to be considered. Producing all possible symmetries a priori is computationally explosive. An alternative way is to filter out these goals which are "rather similar". Using the above definitions, we can note, for example, the equivalent goals of order 1, and put them aside.

If proofs for two such goals are required at a later stage, we might either check whether they are indeed symmetric or prove one of the goals and then try to apply it to the other one, and even modify it if it almost works but not quite.

BIBLIOGRAPHY

Eubank, Porter. "The Geometry Strategist: a Program for Proving Geometry Theorems." Unpublished paper. August 30, 1972.

Gelernter, H. "A Note on Syntactic Symmetry and the Manipulation of Formal System by Machine." Information and Control 2 (1961), 80-89.

Gelernter, H. "Realization of Geometry-Theorem Proving Machine." In Feigenbaum and Feldman (eds.), COMPUTERS AND THOUGT, 134-152.

Gelernter, H. et. al. "Empirical Exploration of the Geometry-Theorem Proving Machine." In COMPUTERS AND THOUGHT, 153-163.

Goldstein, I. "Elementary Geometry Theorem Proving." A.I. Memo No. 280, M.I.T April 1973.

Nevins, A. J. "Plane Geometry Theorem Proving Using Forward Chaining" A.I. Memo No. 303 M.I.T January 1974.

Wong, R. "Construction Heuristics for Geometry and a Vector Algebra Representation of geometry." Project MAC technical memorandum 28, M.I.T, June 1972.